OpenGL ES Shading and Rendering

Session 319

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

OpenGL ES Shading and Rendering

Luc Semeria and Alex Eddy
Software Engineers
Embedded Graphics Acceleration

Agenda

Shading

- OpenGL ES 2.0 programmable graphics pipeline
- Writing shaders
- Working with shaders

Advanced Rendering Techniques

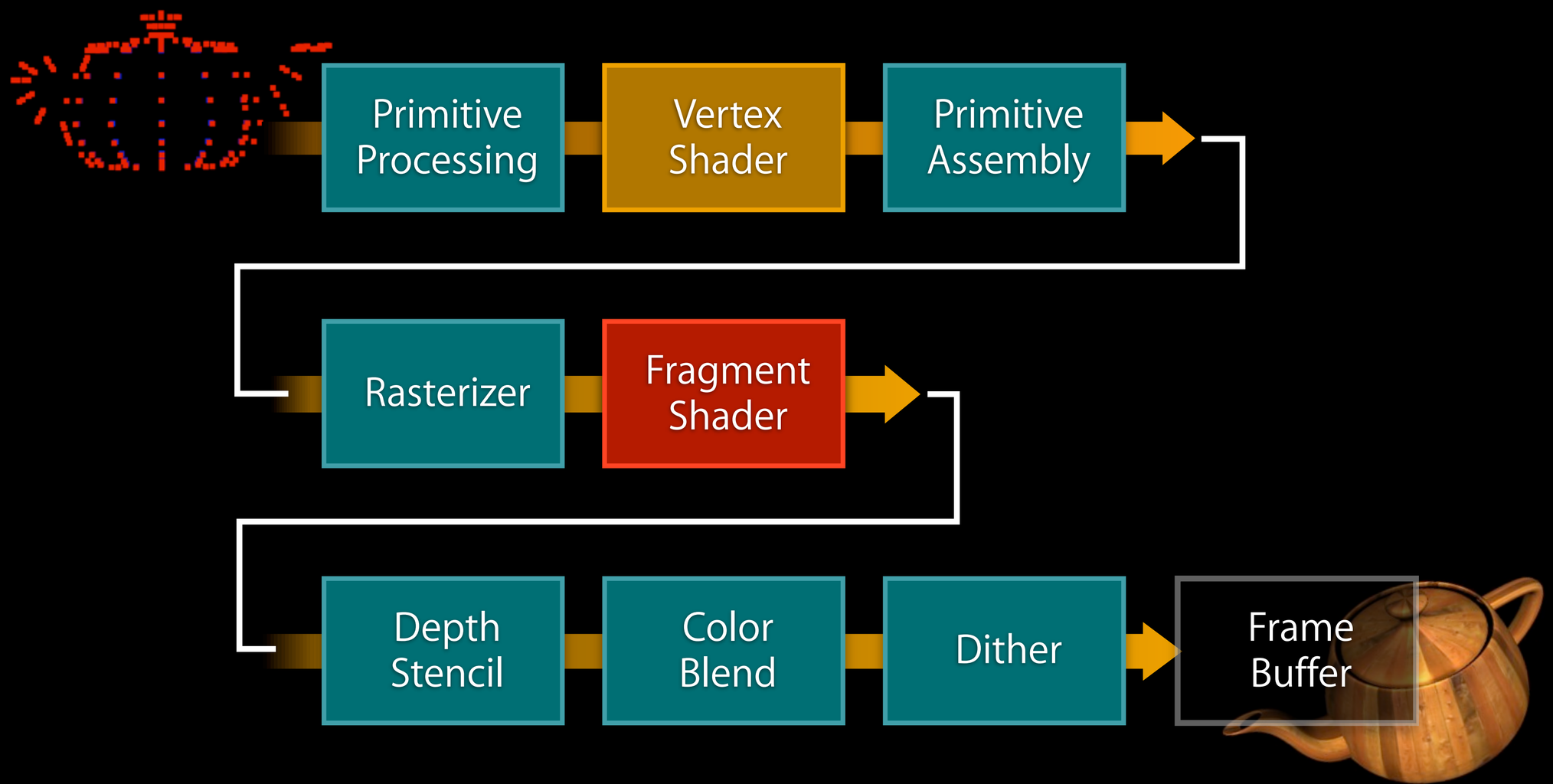
- 3D effects
- 2D image processing
- Combining 2D and 3D

Agenda

- OpenGL ES 2.0 programmable graphics pipeline
- Writing shaders
- Working with shaders

OpenGL ES 2.0

Programmable graphics pipeline



OpenGL ES 2.0

Vertex and Fragment Shaders

```
void main(void)
{
  gl_Position = ...
}
```

Vertex Shader

```
void main(void)
{
  gl_FragColor = ...
}
```

Fragment Shader

Program

Programmable Graphics Capabilities

- Tangent space bump map
- Cubic environment map
- Refraction
- Better image processing
- Much more!

Agenda

- OpenGL ES 2.0 programmable graphics pipeline
- Writing shaders
- Working with shaders

Vertex Shader

Replaces OpenGL ES 1.1 fixed-function vertex units

- Position and normal transformation
- Texture coordinate transformation
- Lighting equation

Vertex Shader

Inputs

- Attributes

  Per vertex data (coordinates, normals)

- Uniforms

  Constant data (matrix-transform, light coordinates)

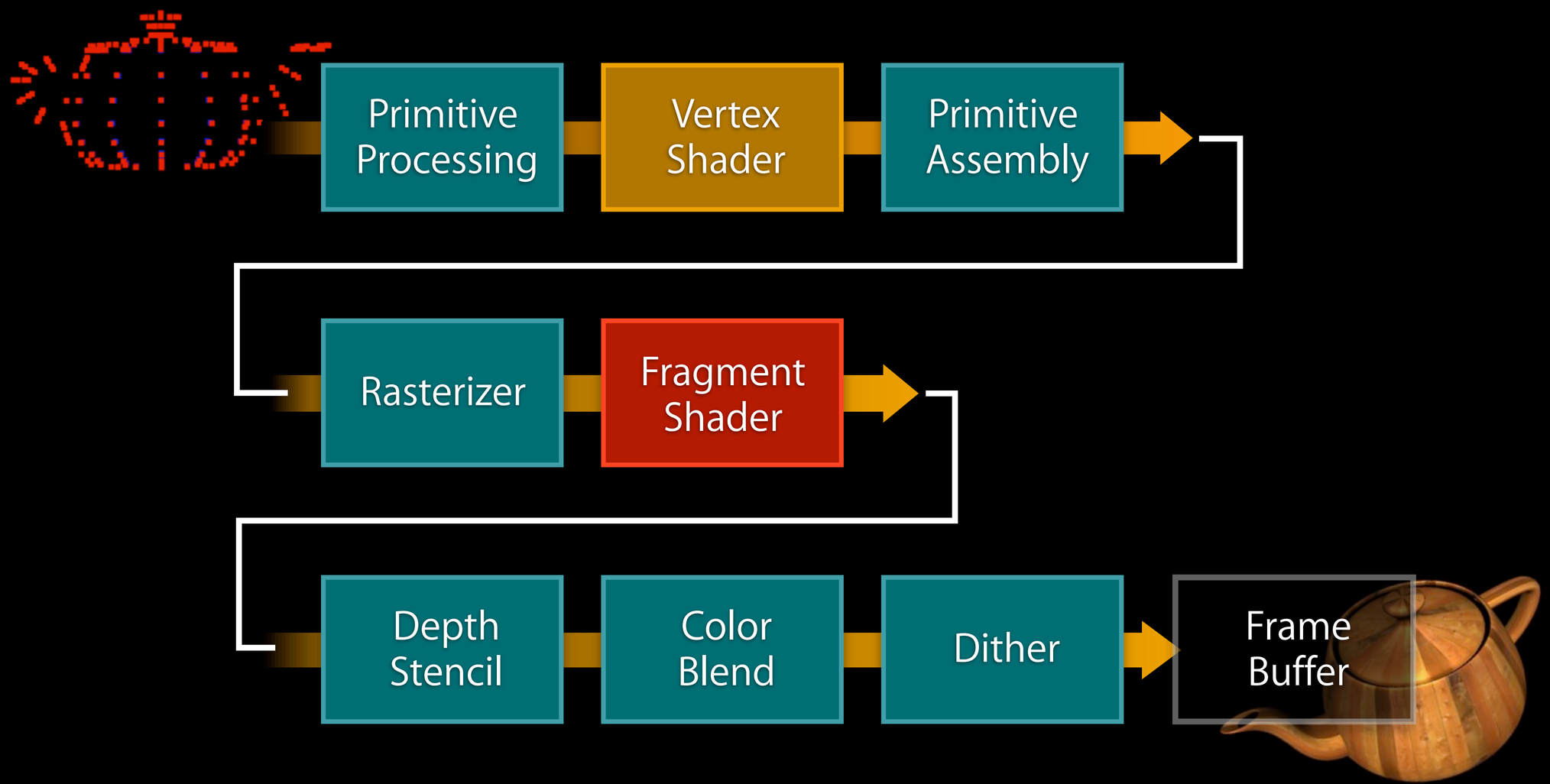
Outputs

- Varyings

  Per vertex output variables (position, texture coordinates)

OpenGL ES 2.0

Programmable graphics pipeline

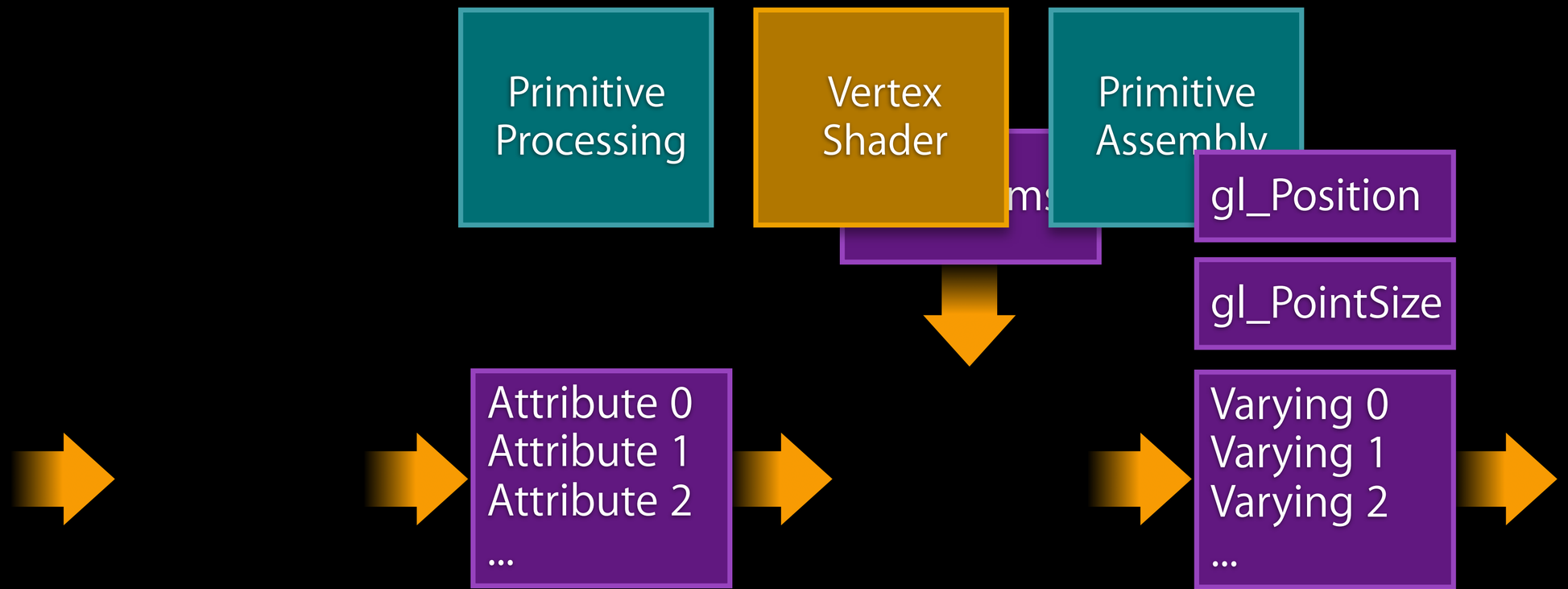


Vertex Shader I/O

Attributes, Uniforms and Varyings

Vertex Shader I/O

Attributes, Uniforms and Varyings



Fragment Shader

Replaces OpenGL ES 1.1 fixed function fragment units

- Texture loading
- Texture environment
- Fog
- Alpha test

Fragment Shader

Inputs

- Varyings

  Interpolated vertex output (position, texture coordinates, color)

- Uniforms

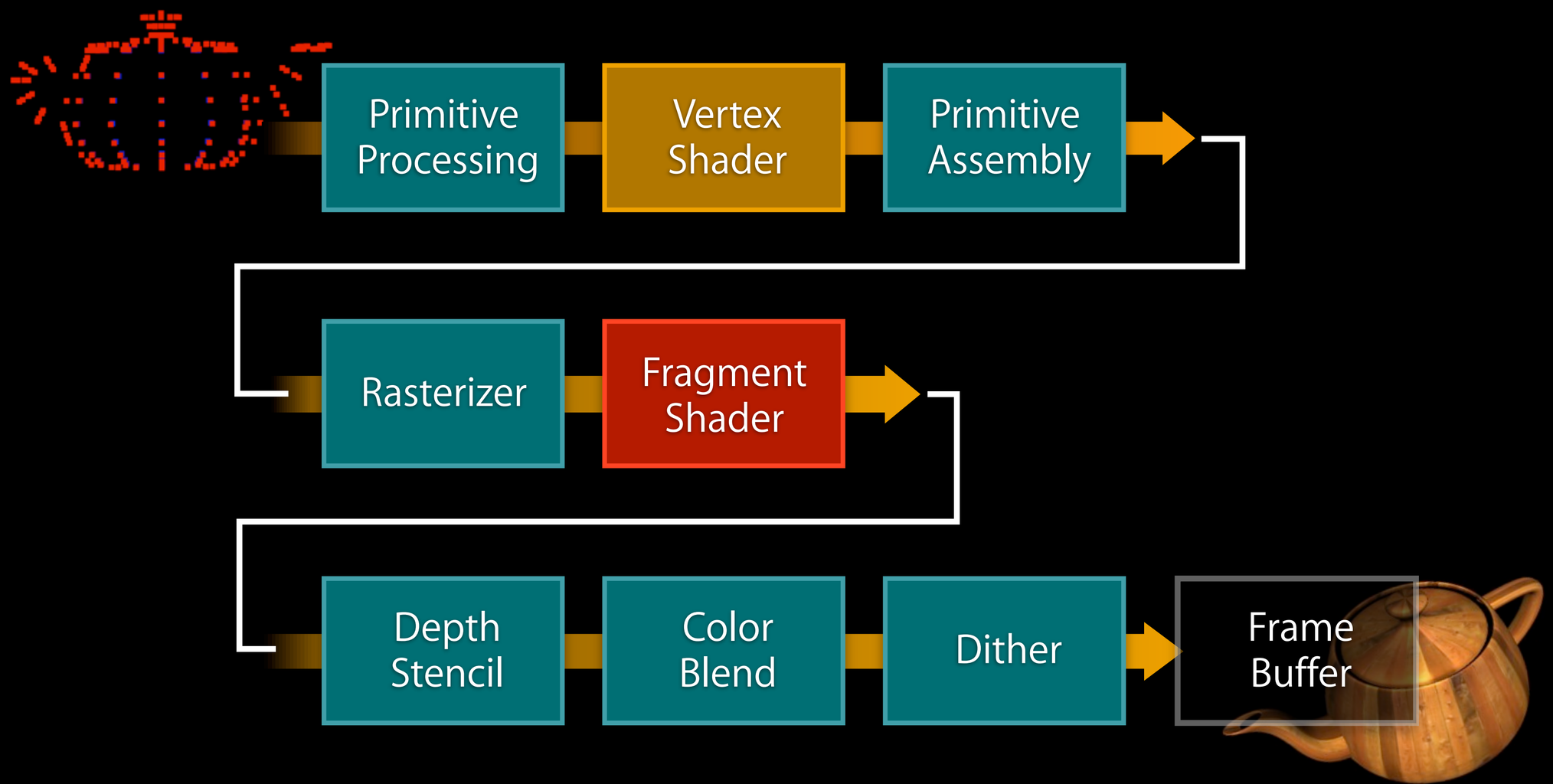
  Constant data (fog characteristics, texture unit)

Output

- Fragment color
- Optional discard

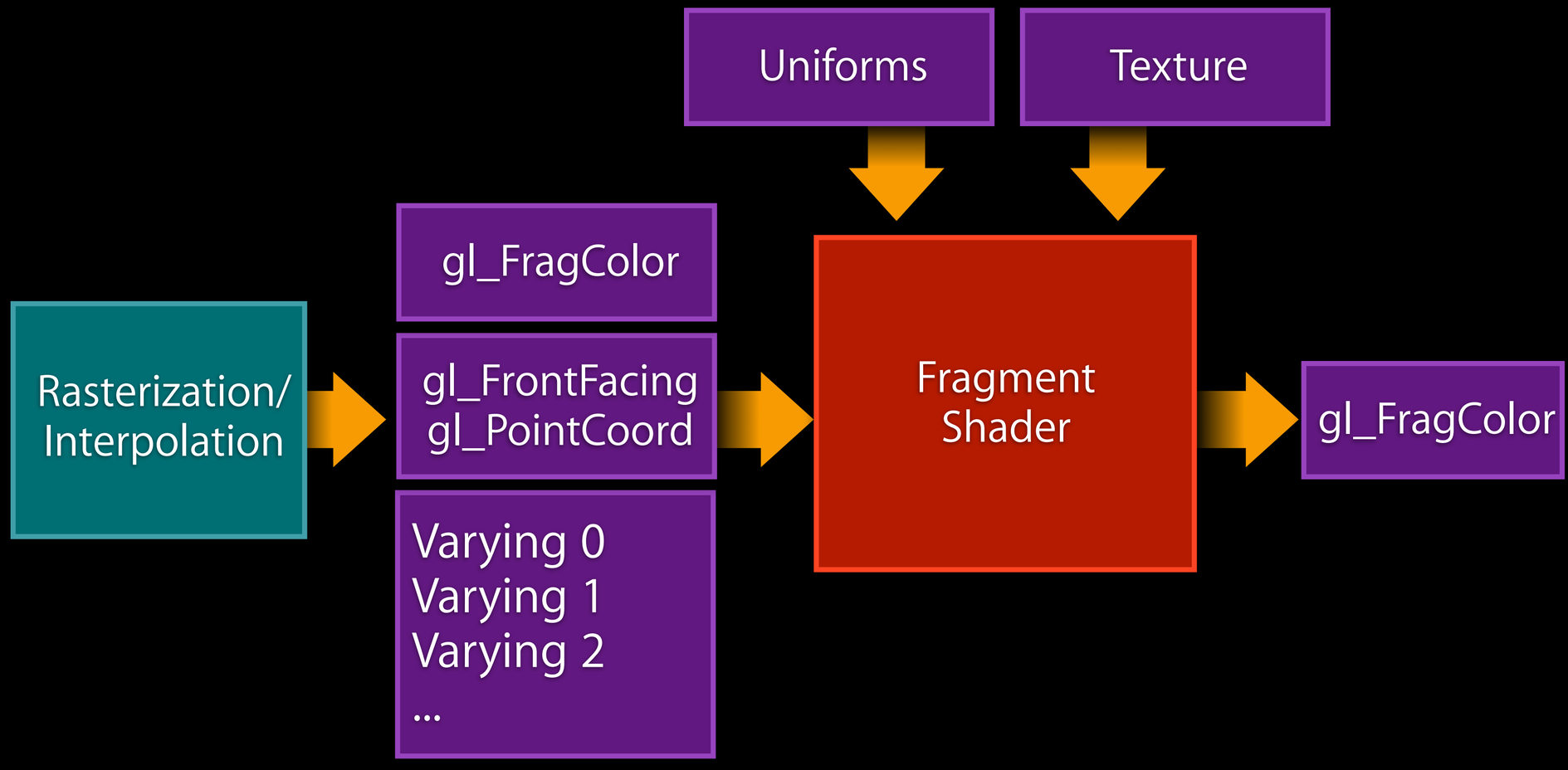
OpenGL ES 2.0

Programmable graphics pipeline



Fragment Shader I/O

Varyings, Uniforms, FragColor



OpenGL ES Shading Language

- GLSL: C-like language
  - Procedural (main function)
  - Compiled, linked
- Limitations
  - No pointers, gotos, recursions
- Extensions
  - Storage
  - Data types, precision
  - Built-ins

Shader Program Example

Render model with constant color

Fragment Shader

```
void main(void)
{
  gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
}
```

Shader Program Example

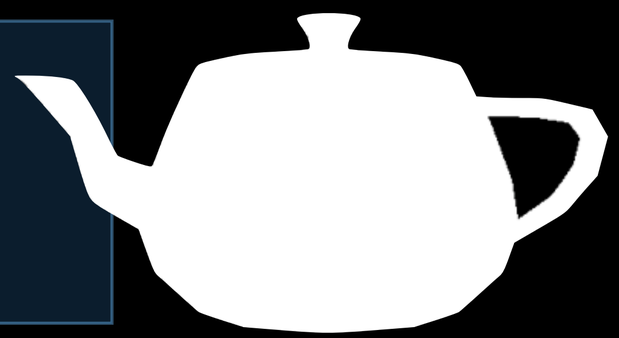
Render model with constant color

Vertex Shader

```
attribute vec4 position;
uniform mat4 modelViewProjectionMatrix;

void main(void)
{
  gl_Position = modelViewProjectionMatrix * position;
}
```

Fragment Shader

```
void main(void)
{
  gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
}
```



GLSL Storage Qualifiers

```
attribute vec4 position;
uniform mat4 modelViewProjectionMatrix;
```

`attribute`	per vertex attribute input
`uniform`	uniform input
`varying`	per vertex output/ per fragment input
`const`	compile time constant

GLSL Datatypes and Operations

Basic	`float, int, bool`

Vector	`vec2, vec3, vec4`
	`ivec2, ivec3, ivec4`
	`bvec2, bvec3, bvec4`

Matrix	`mat2, mat3, mat4`
Sampler	`sampler2D, samplerCube`
Array	`vec4 v[5];`
Structure	`struct complex { float r; float i; };` `complex c;`

Arithmeric and Logic Operators

`+ - * /`

GLSL ES Built-in Functions

Angle and trigonometry	`sin, cos, atan, degrees,...`
Exponential	`pow, sqrt, inversesqrt, exp2,...`
Common	`abs, floor, mod, min, max, clamp, mix, step,...`
Geometric	`length, dot, cross, normalize, reflect, refract,...`
Matrix	`matrixCompMult`
Relational	`equal, any, all,...`
Texture Lookup	`texture2D, textureCube,...`
Derivative Functions (gl_standard_derivative_OES)	`dFdx, dFdy, fwidth`

GLSL ES Built-in Functions

Angle and trigonometry	sin, cos, atan, degrees,...
Exponential	pow, sqrt, inversesqrt, exp2,...
Common	abs, floor, mod, min, max, clamp, mix, step,...
Geometric	length, dot, cross, normalize, reflect, refract,...
Matrix	matrixCompMult
Relational	equal, any, all,...
Texture Lookup	texture2D, textureCube,...
Derivative Functions (gl_standard_derivative_OES)	dFdx, dFdy, fwidth

Vertex Shader for Texture Mapping

Matrix transformation

Vertex Shader

```
attribute vec4 position;
attribute vec2 texCoord;

uniform mat4 modelViewProjectionMatrix;

varying vec2 texCoordVarying;

void main(void)
{
  gl_Position = modelViewProjectionMatrix * position;
  texCoordVarying = texCoord;
}
```

Fragment Shader for Texture Mapping

Texture load

Fragment Shader

```
uniform sampler2D texture1, texture2;

varying highp vec2 texCoordVarying;

void main(void)
{
  lowp vec4 color1, color2;

  color1 = texture2D(texture1, texCoordVarying);
  color2 = texture2D(texture2, texCoordVarying);
  gl_FragColor = color1 + color2;
}
```

GLSL ES Precision Qualifier

```
varying highp vec2 texCoordVarying;
...
lowp vec4 color1, color2;
```

Type qualifier

`lowp`	`mediump`	`highp`

- GLSL ES compiler hint
- Reduce range and precision (color)

Default Precision

Predefined default precision

	vertex shader	fragment shader
float	highp	undefined

- Precision must be set for all float variables in fragment shader

Default Precision

User-defined default precision

```
precision highp float;
```

- Float precision applies to float vectors, matrices, etc.

Default Precision in Fragment Shader

Texture load

Fragment Shader

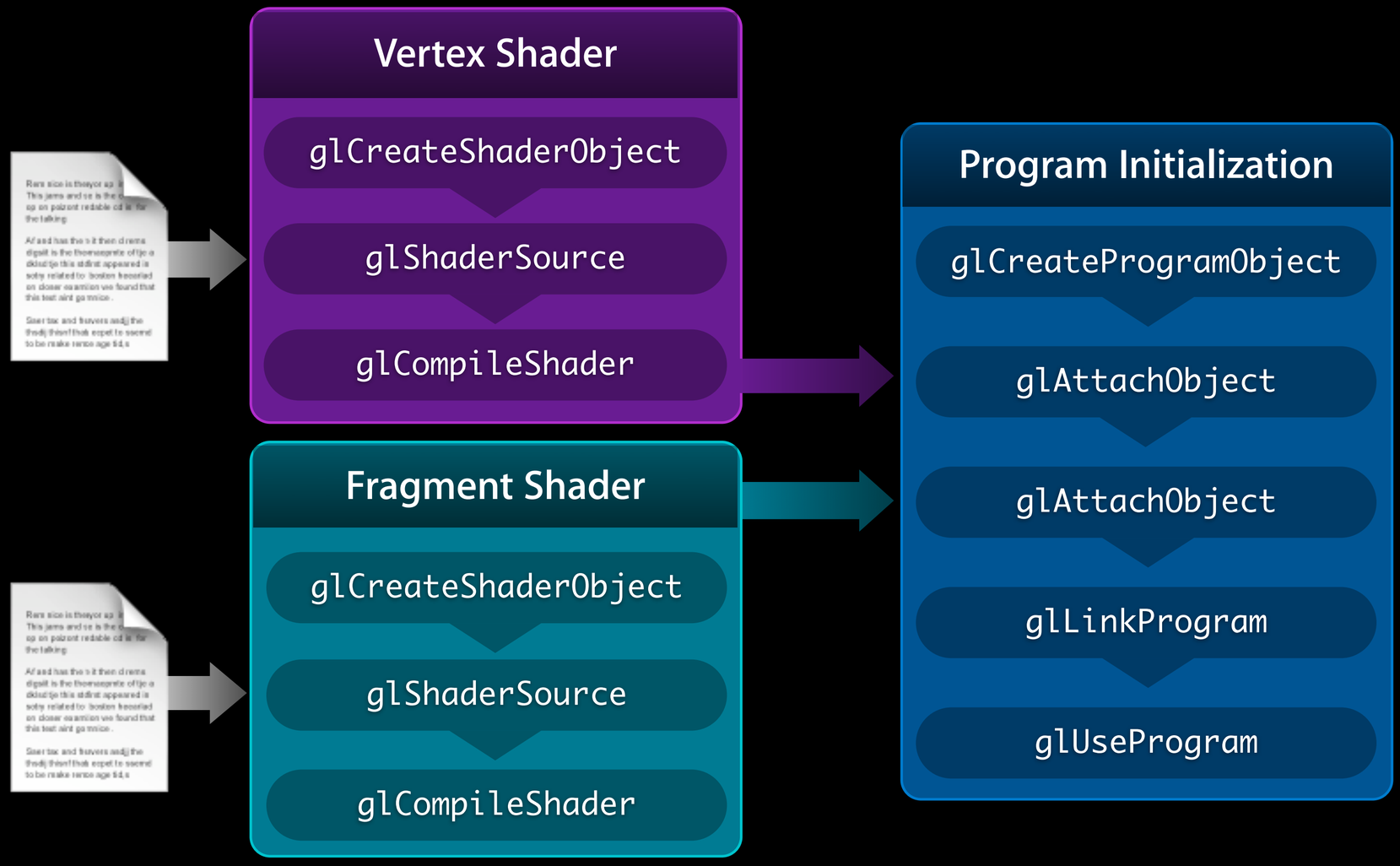
```
#ifdef GL_ES
precision highp float;
#endif

uniform sampler2D texture;

varying vec2 texCoordVarying;

void main(void)
{
  lowp vec4 color1, color2;

  color1 = texture2D(texture1, texCoordVarying);
  color2 = texture2D(texture2, texCoordVarying);
  gl_FragColor = color1 + color2;
}
```

Agenda

- OpenGL ES 2 programmable graphics pipeline
- Writing shaders
- Working with shaders

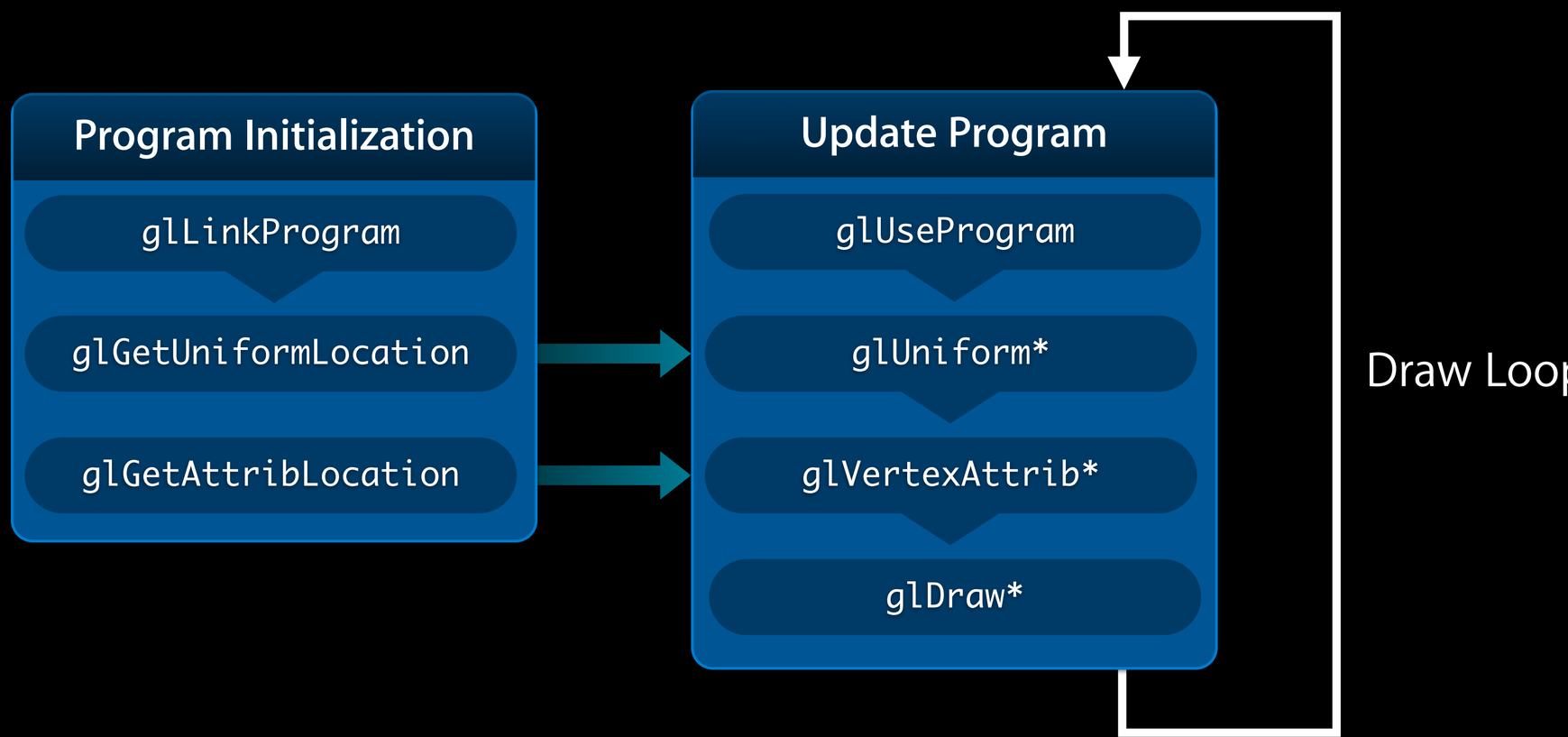
Shader Initial Setup

Compile, Attach, Link, Use



Using Shaders

Set Attributes and Uniform Values



Demo

Diffuse Lighting

Agenda

Shading

- OpenGL ES 2.0 programmable graphics pipeline
- Writing shaders
- Working with shaders

Advanced rendering techniques

- 3D effects
- 2D image processing
- Combining 2D and 3D

Agenda

Shading

- OpenGL ES 2.0 programmable graphics pipeline
- Writing shaders
- Working with shaders

Advanced rendering techniques

- 3D effects
- 2D image processing
- Combining 2D and 3D

- Using the hardware capabilities
  - ES1.1 fixed function
  - ES2.0 shaders

Beyond Texturing and Blending

- Make better looking games!
- OpenGL is more than flat textured quads
- Use hardware acceleration to its fullest

Beyond Texturing and Blending

Take advantage of hardware capabilities

- Multiple attributes
- Multiple textures
- ES1.1
  - Texture matrix
  - Texture environment combine
- ES2.0
  - Shaders

Demo

3D Effects

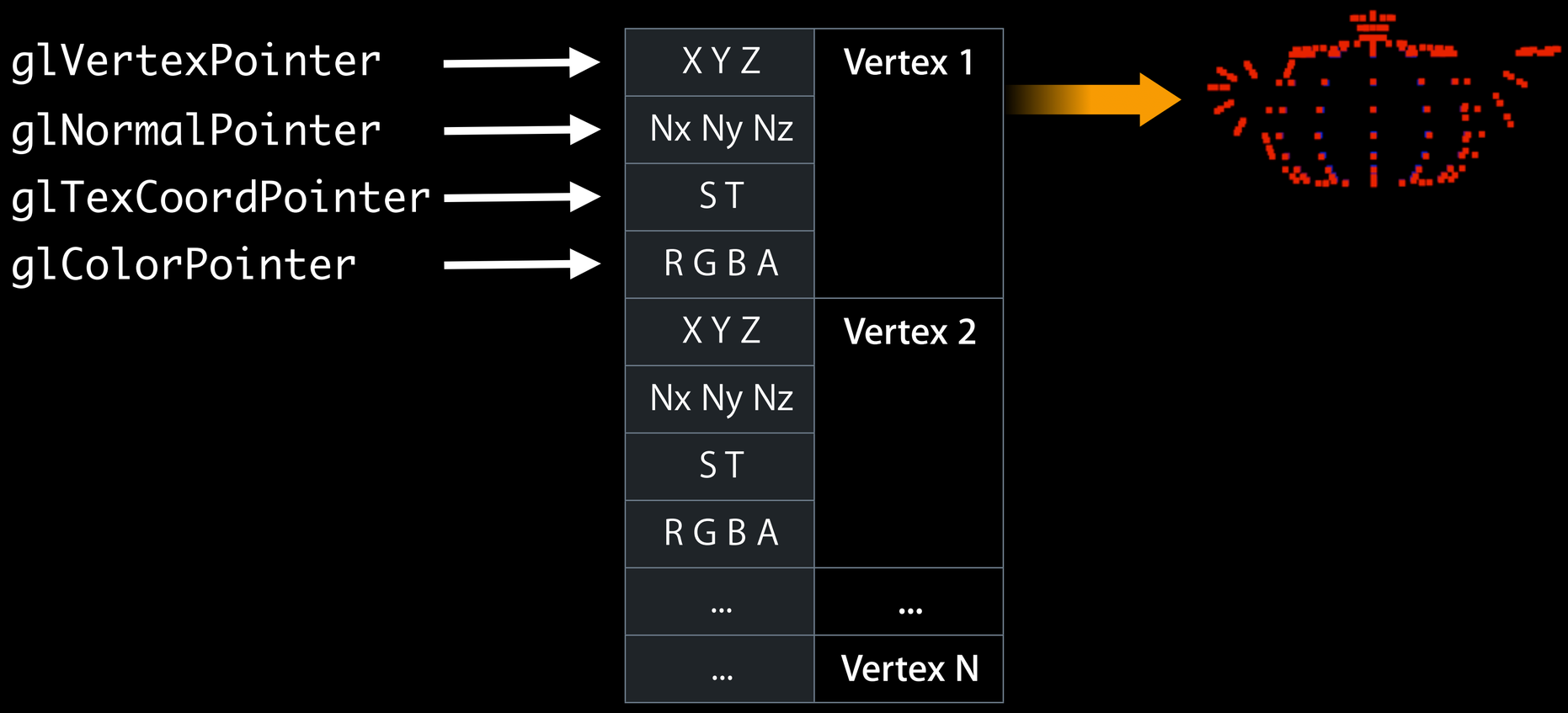
How Can I Do That?

- Know thy pipeline
  - Learn all the knobs and switches
- Define your problem in GL terms
  - ES1.1: TexCoord, Texture Matrix, TexEnv
  - ES2.0: GLSL shader
- Break large problems into smaller pieces
  - Multipass and Render-to-Texture

Know Thy Pipeline

Vertex attributes

- Vertex Arrays contain attributes
  - i.e. interleaved in a static VBO

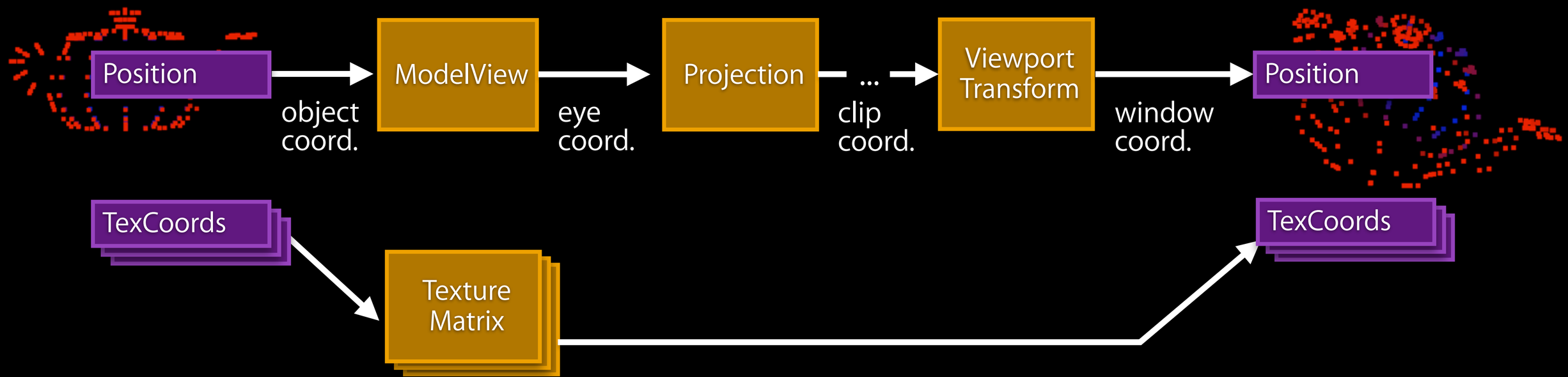


Know Thy Pipeline

Vertex transformation

Vertex Processing

- Object coordinates transformed to window coordinates
  - Scale, Rotate, Translate, Project…



- Texture Matrices, too!

Know Thy Pipeline

ES1.1 Texture Matrix

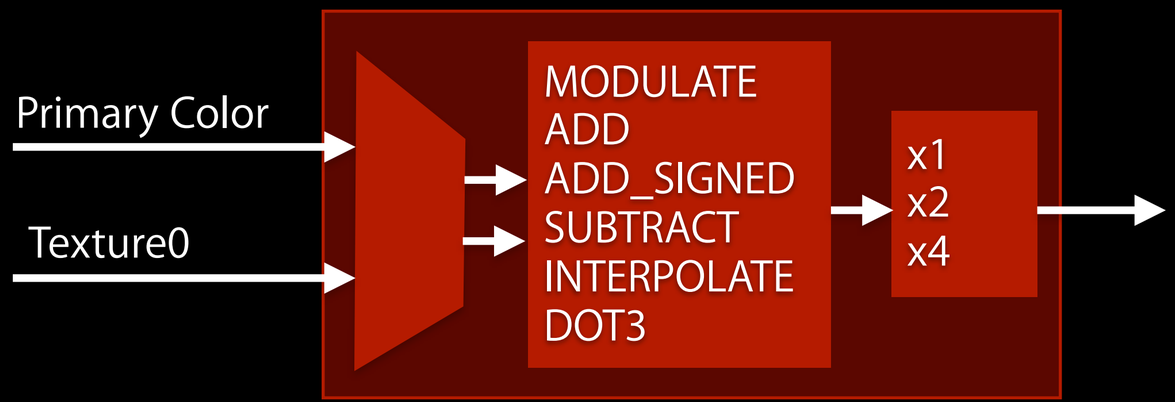
- Transform texture coordinates
- Example
  - Rotate textured sphere?
  - Rotate texture around sphere!

Know Thy Pipeline

ES1.1 Texture Combine

Texture Environment

- TexEnv Combine
  - Simple math operations per-fragment (A*B+C)
  - All inputs and outputs clamped to [0, 1]

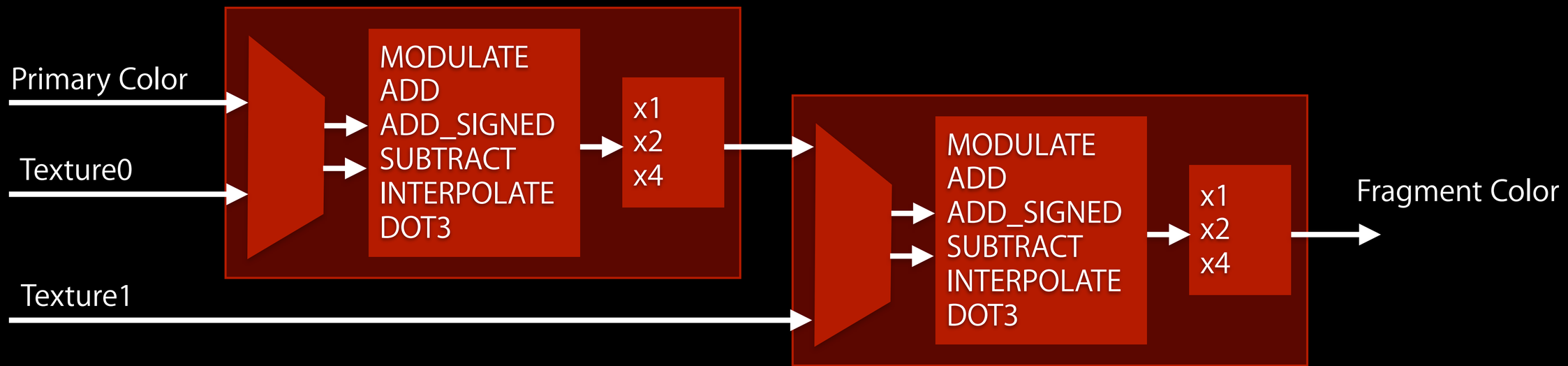


Know Thy Pipeline

ES1.1 Texture Combine

Texture Environment

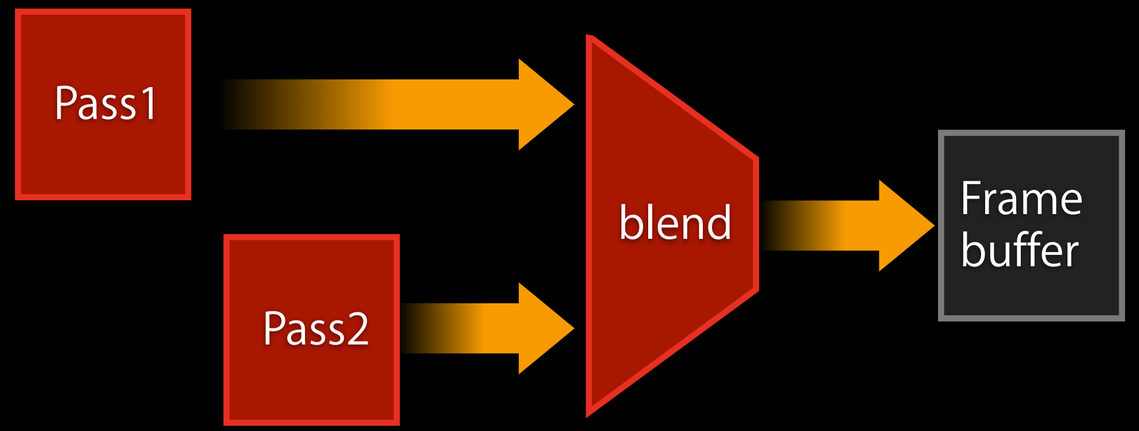
- Multitexturing
  - Two texture units on MBX
  - Eight texture units on SGX



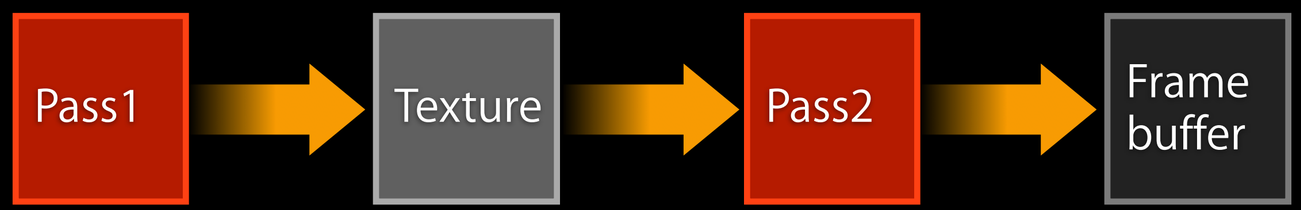
Multipass and Render-to-Texture

Break large problems into smaller pieces

Multipass with framebuffer blending



Multipass with render to texture



Know Thy Pipeline

ES1.1 vs. ES2.0

- ES1.1
  - A few fixed purpose attributes (glTexCoordPointer etc)
  - Fixed function transformation (texture matrix)
  - TexEnv limited to simple math operations (add/multiply)
  - Multiple passes often required
- ES2.0
  - 16 generic attributes (glVertexAttribPointer)
  - Programmable vertex and fragment shaders

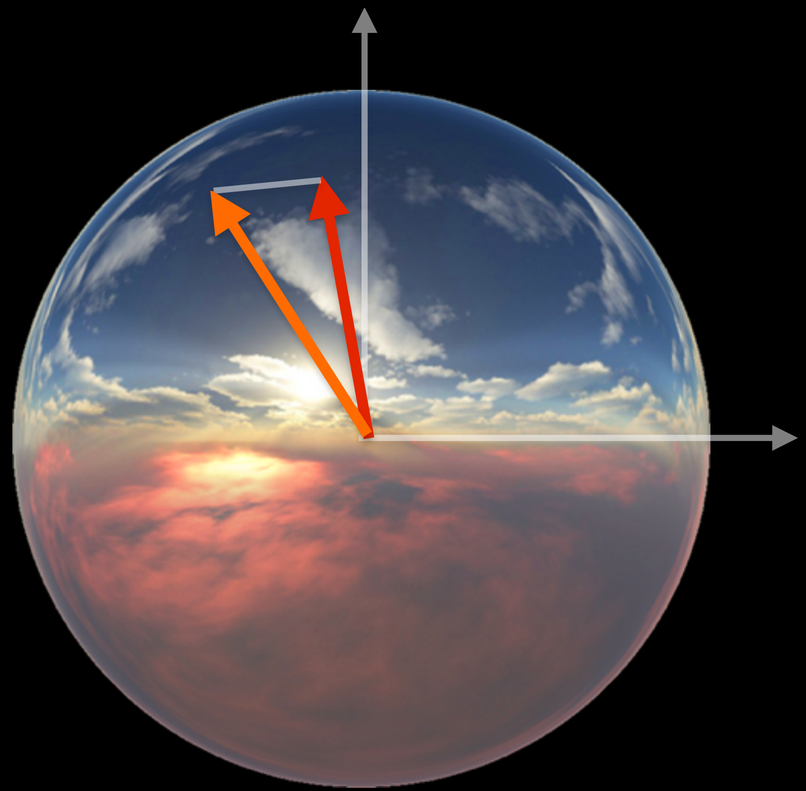
Environment Mapping

Putting it together

Environment Mapping

Concept

- Vertex normal (Nx, Ny, Nz) in eye space
- Lookup color in environment map at (Nx, Ny)



Environment Mapping

Use GLSL in ES2.0

Vertex Shader

```
attribute vec3 position, normal;
attribute vec4 color;

uniform mat4 myEnvMatrix, myModelViewProjectionMatrix;

varying vec2 texCoord;
varying vec4 colorVarying;

void main(void)
{
  texCoord = myEnvMatrix * normal;
  colorVarying = color;
  gl_Position = myModelViewProjectionMatrix * position;
}
```

Environment Mapping

Use GLSL in ES2.0

Fragment Shader

```
uniform sampler2D unit0;

varying highp vec2 texCoord;
varying lowp  vec4 colorVarying;

void main (void)
{
  lowp vec4 sample;

  sample        = texture2D(unit0, texCoord);
  gl_FragColor = sample;
}
```

Environment Mapping

Use GLSL in ES2.0

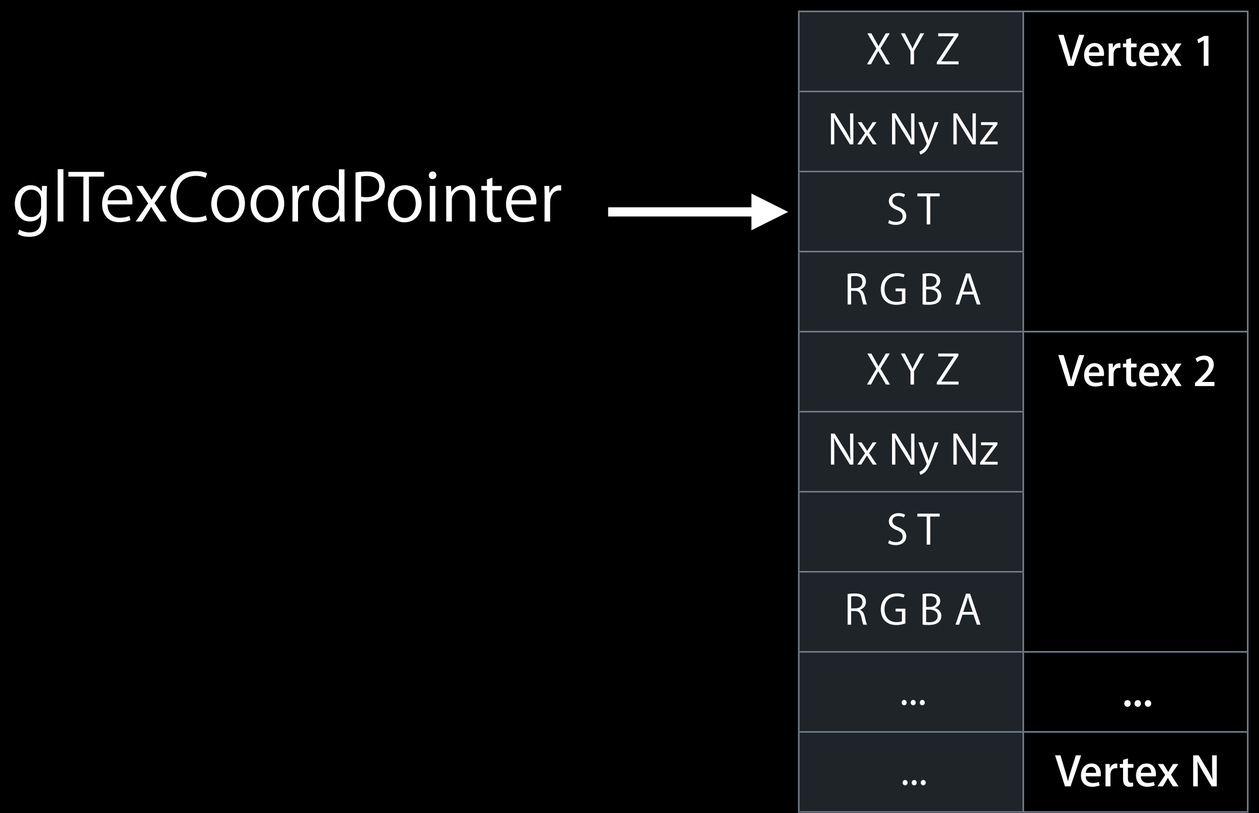
Fragment Shader

```
uniform sampler2D unit0;

varying highp vec2 texCoord;
varying lowp  vec4 colorVarying;

void main (void)
{
  lowp vec4 sample;

  sample        = texture2D(unit0, texCoord);
  gl_FragColor = mix(sample.rgb, colorVarying, sample.a); // Fresnel
}
```

Environment Mapping

Use ES1.1 attributes

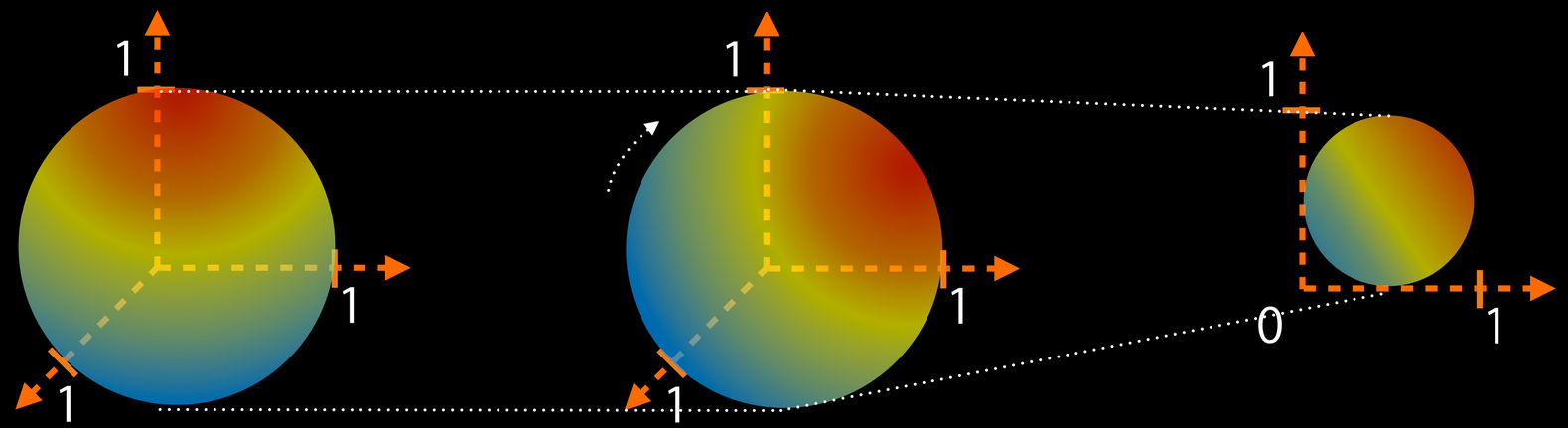
- Use Normals as TexCoords



Environment Mapping

Use the ES1.1 Texture Matrix

- Use Texture Matrix
  - Transform from object space to eye space
  - Transform from [-1, 1] normal to [0, 1] texture coordinate



3D normal space

2D texture coordinate

Environment Mapping

Using Texture Matrix in ES1.1

```
// unit 0 uses normal.XYZ to reflect spheremap
glMatrixMode(GL_TEXTURE);
glLoadIdentity();

// normals are in [-1..1], scale to [0..1]
glTranslatef(0.5, 0.5, 0.5);
glScalef    (0.5, 0.5, 0.5);
```

```
// transform normals by the modelview matrix
glMultMatrixf(myModelViewMatrix);

// set drawing state
glBindTexture(GL_TEXTURE_2D, TEXTURE_ENVMAP);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);




// draw
myDrawGeometry(geom[object]);
```

Environment Mapping

Using Texture Combine in ES1.1 to Interpolate by Fresnel Term

```
// unit 0 uses normal.XYZ to reflect spheremap
glMatrixMode(GL_TEXTURE);
glLoadIdentity();

// normals are in [-1..1], scale to [0..1]
glTranslatef(0.5, 0.5, 0.5);
glScalef(0.5, -0.5, 0.5);

// transform normals by the modelview matrix
glMultMatrixf(myModelViewMatrix);

// set drawing state
glBindTexture(GL_TEXTURE_2D, TEXTURE_ENVMAP);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_COMBINE);
glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_RGB, GL_INTERPOLATE);
glTexEnvi(GL_TEXTURE_ENV, GL_SRC0_RGB, GL_TEXTURE);       // RGB
glTexEnvi(GL_TEXTURE_ENV, GL_SRC1_RGB, GL_PRIMARY_COLOR); // RGB
glTexEnvi(GL_TEXTURE_ENV, GL_SRC2_RGB, GL_TEXTURE);       // A

// draw
myDrawGeometry(geom[object]);
```

Demo

Environment mapping and beyond

More 3D effects

Be creative

- Many more effects are possible in ES1.1
  - Toon shading
  - Anisotropic Lighting
  - Gloss mapping
  - Object space bump mapping
- ES2.0 allows infinitely more
  - Dependent texture lookups (refraction)
  - Branching
  - Complex math

Image Processing

Image Processing

Greyscale in ES2

- Simple filter
  - Average Grey = R*0.33 + G*0.33 + B*0.33

Fragment Shader

```
uniform   sampler2D unit0;
varying   mediump vec2 tex;
const     lowp vec3 weights = {0.33, 0.33, 0.33};

void main (void)
{
  lowp vec4 sample = texture2D(unit0, tex);
  gl_FragColor = vec4(vec3(dot(weights, sample.rgb)), sample.a);
}
```

Image Processing

Greyscale in ES1.1

- Texture combine DOT3 needs input biased and scaled to [0.5,1.0]

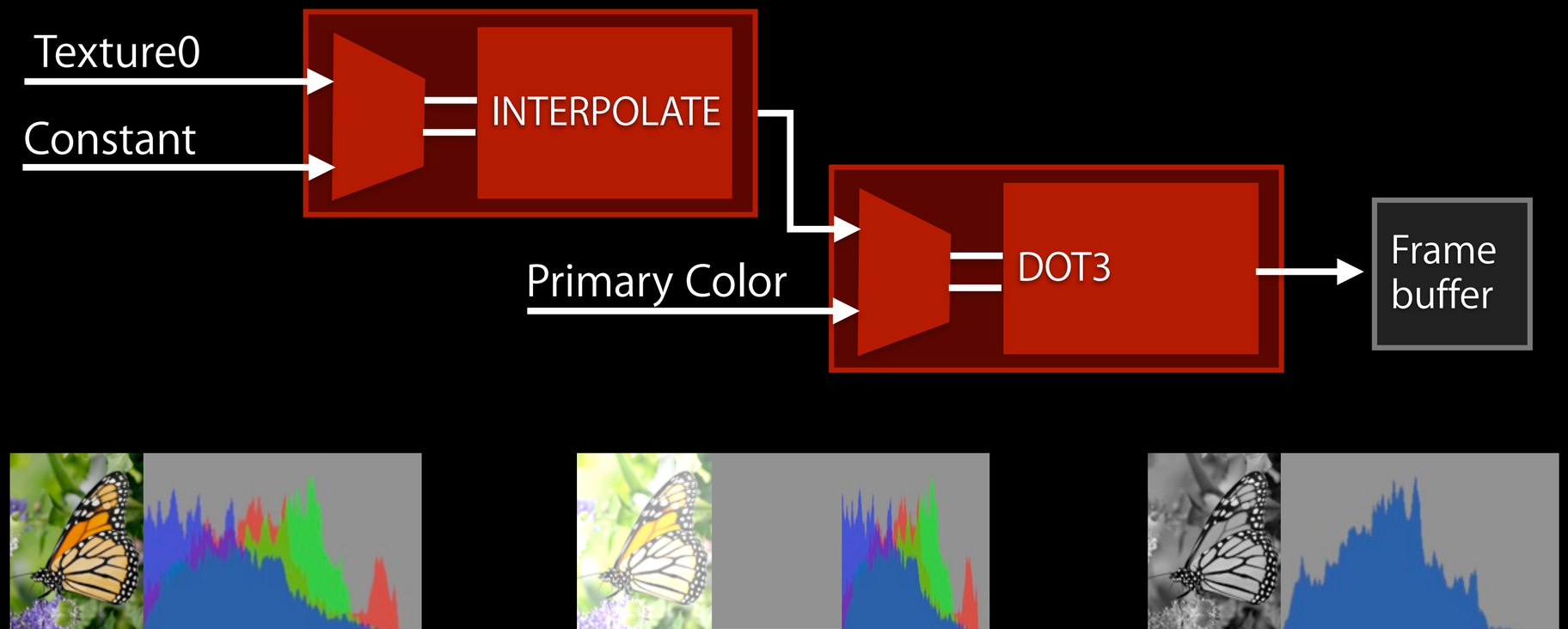


Image Processing

Greyscale in ES1.1

```
GLfloat weights[4] = { .667, .667, .667, 0.5 };  // average, pre-biased to [0.5..1]
GLfloat lerp[4]    = { 1.0, 1.0, 1.0, 0.5 };    // interpolate towards white

// Unit 0 scales and biases into [0.5..1.0]
glActiveTexture(GL_TEXTURE0);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_COMBINE);
glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_RGB,      GL_INTERPOLATE);
glTexEnvi(GL_TEXTURE_ENV, GL_SRC0_RGB,         GL_TEXTURE);
glTexEnvi(GL_TEXTURE_ENV, GL_SRC1_RGB,         GL_CONSTANT);  // RGB
glTexEnvi(GL_TEXTURE_ENV, GL_SRC2_RGB,         GL_CONSTANT);  // A
glTexEnvfv(GL_TEXTURE_ENV,GL_TEXTURE_ENV_COLOR, lerp);

// Unit 1 dot products with weights
glActiveTexture(GL_TEXTURE1);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_COMBINE);
glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_RGB,      GL_DOT3_RGB);
glTexEnvi(GL_TEXTURE_ENV, GL_SRC0_RGB,         GL_PREVIOUS);
glTexEnvi(GL_TEXTURE_ENV, GL_SRC1_RGB,         GL_PRIMARY_COLOR);

// Set primary color and draw
glColor4f(weights[0], weights[1], weights[2], weights[3]);
glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
```

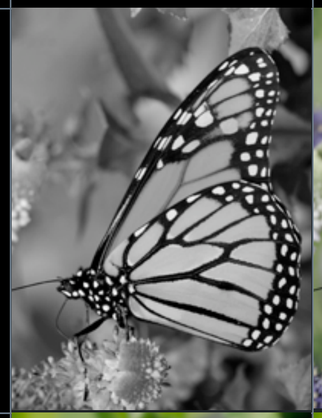
Image Processing

Greyscale in ES1.1

```
GLfloat weights[4] = { .667, .667, .667, 0.5 };  // average, pre-biased to [0.5..1]
GLfloat lerp[4]    = { 1.0, 1.0, 1.0, 0.5 };    // interpolate towards white

// Unit 0 scales and biases into [0.5..1.0]
glActiveTexture(GL_TEXTURE0);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_COMBINE);
glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_RGB,      GL_INTERPOLATE);
glTexEnvi(GL_TEXTURE_ENV, GL_SRC0_RGB,         GL_TEXTURE);
glTexEnvi(GL_TEXTURE_ENV, GL_SRC1_RGB,         GL_CONSTANT);  // RGB
glTexEnvi(GL_TEXTURE_ENV, GL_SRC2_RGB,         GL_CONSTANT);  // A
glTexEnvfv(GL_TEXTURE_ENV,GL_TEXTURE_ENV_COLOR, lerp);

// Unit 1 dot products with weights
glActiveTexture(GL_TEXTURE1);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_COMBINE);
glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_RGB,      GL_DOT3_RGB);
glTexEnvi(GL_TEXTURE_ENV, GL_SRC0_RGB,         GL_PREVIOUS);
glTexEnvi(GL_TEXTURE_ENV, GL_SRC1_RGB,         GL_PRIMARY_COLOR);

// Set primary color and draw
glColor4f(weights[0], weights[1], weights[2], weights[3]);
glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
```

2D Image Processing

- Theory: Image Processing by interpolation and extrapolation
  - http://www.graficaobscura.com/interp/

t	0.0	0.5	1.0	1.5	2.0
Saturation					
Sharpness					

Redefine Your Problem

Fitting inside ES1.1 limitations

- Standard interpolation: ***Src*(t)*** + ***Dst*(1-t)***
  - Maps directly to Texture Combine INTERPOLATE
  - ***Src, Dst,*** and ***t*** must all be in [0..1]
  - How to handle extrapolation by ***t*** > 1?

Redefine Your Problem

Fitting inside ES1.1 limitations

- Standard interpolation: ***Src*(t)** + ***Dst*(1-t)***
  - Maps directly to Texture Combine INTERPOLATE
  - ***Src, Dst,*** and ***t*** must all be in [0..1]
  - How to handle extrapolation by ***t*** > 1?

- Simple Algebra, substitute ***t=2t'***

  Src*(2t') + Dst*(1-2t')

  = 2(Src*(t') + Dst*(0.5-t'))

  - Now ***t'*** is in [0..1]
  - Everything maps to Texture Combine, including final multiply by two

Demo

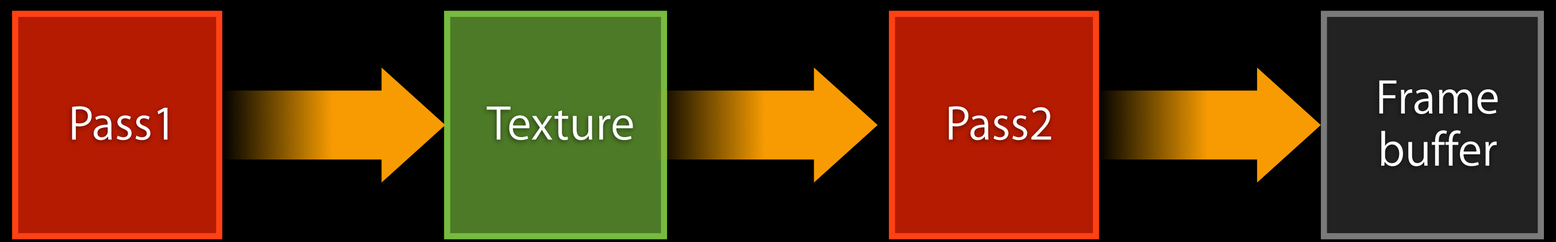
Image Processing

2D and 3D

Fullscreen Postprocessing

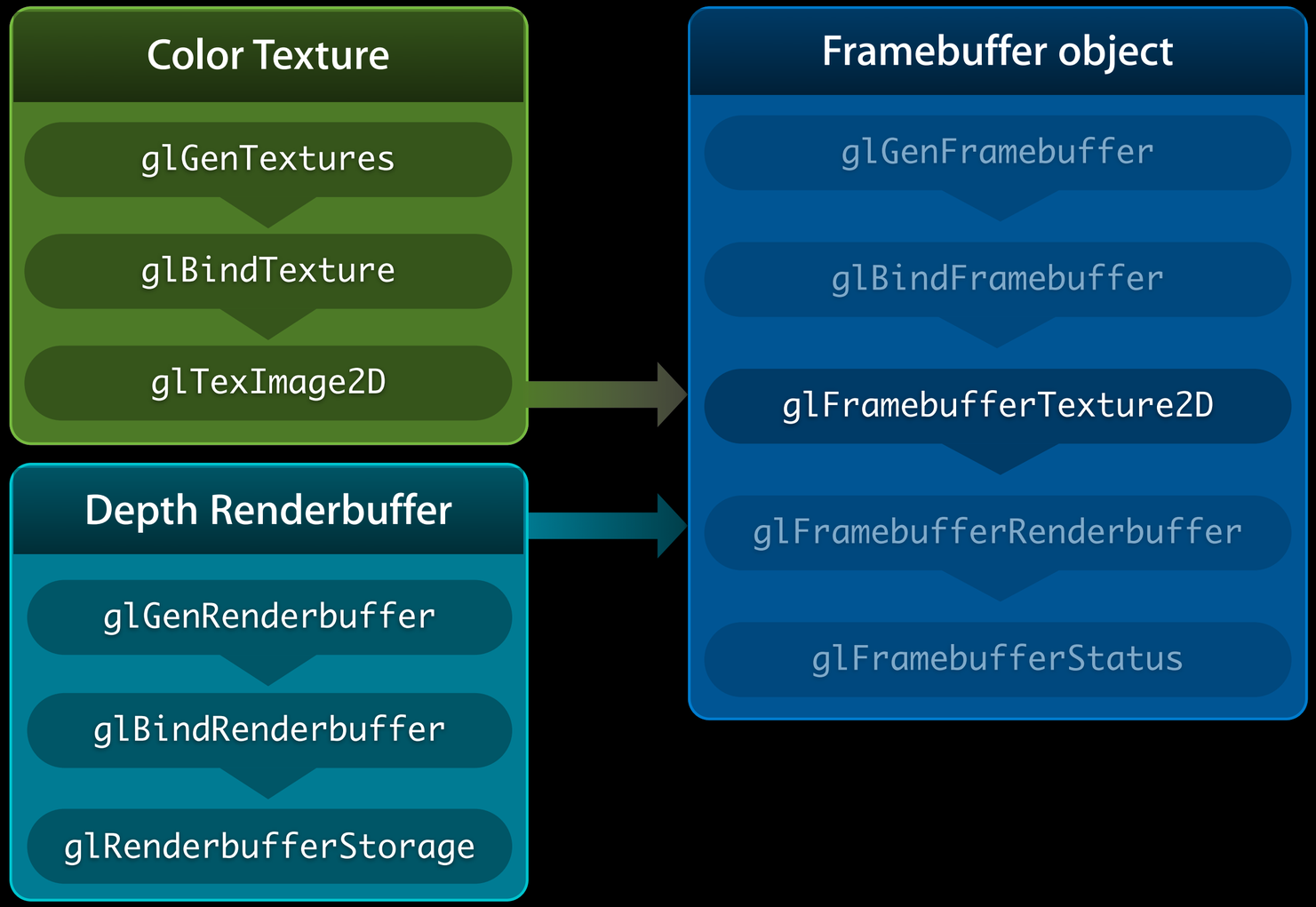
Framebuffer Objects

- Render 3D scene to texture attached to FBO
- Use result as input to additional processing



Setup Render to Texture

Replace color Renderbuffer by texture



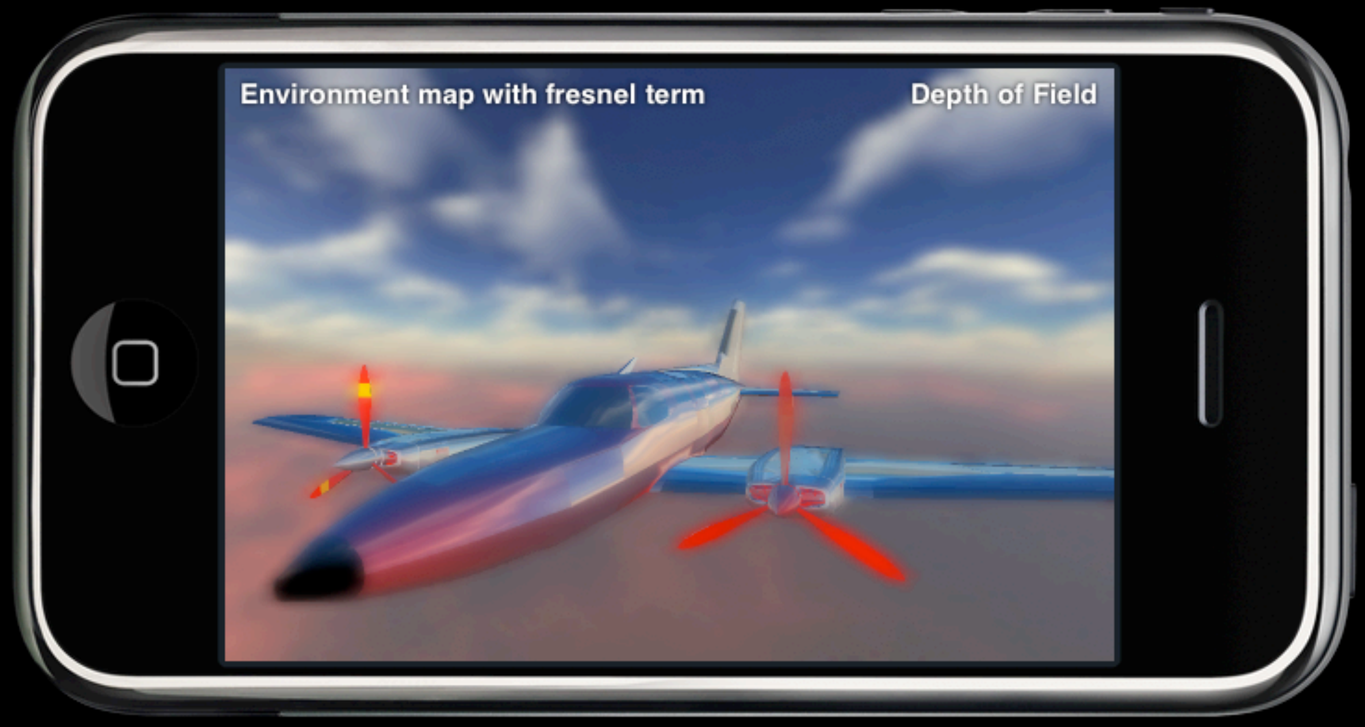
Fullscreen Postprocessing

Bloom effect

- Multipass with render to texture

Fullscreen Postprocessing

- Bloom
- Depth of field
- Blur
- Image processing



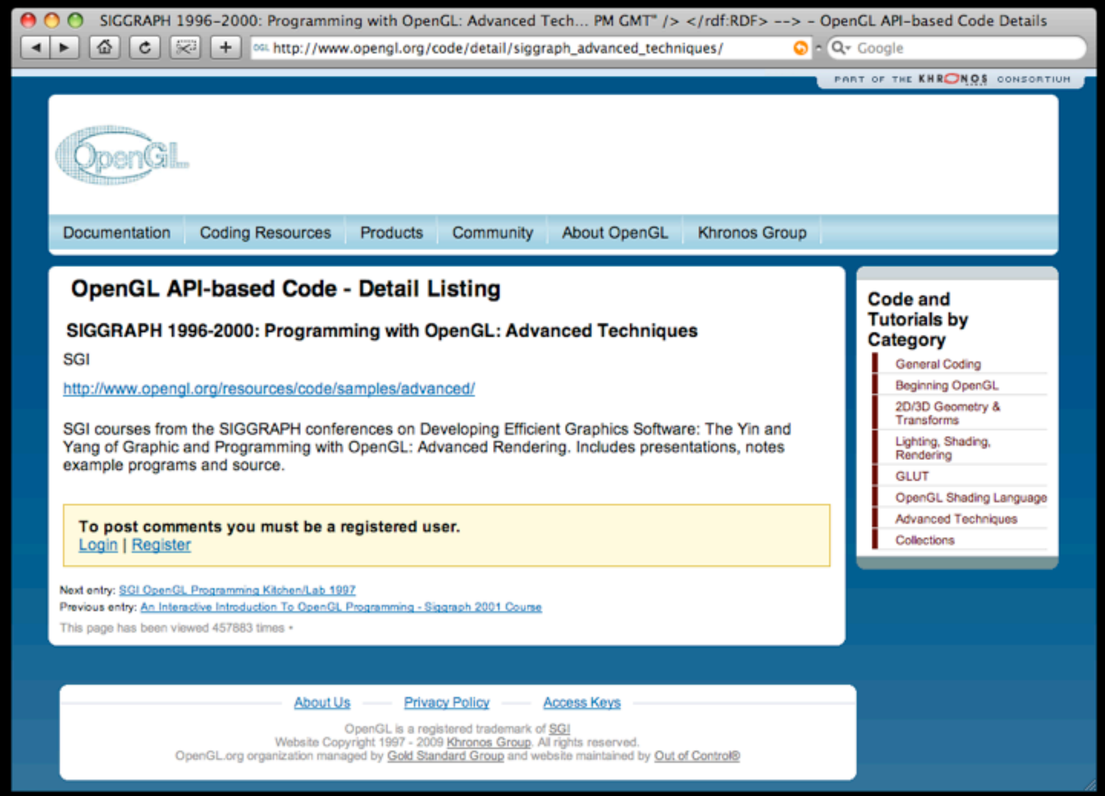
Summary

- Know your OpenGL ES pipeline
- ES 2.0: Programmable shaders
- ES 1.1: Work with fixed functions
  - Texture coordinates and matrix transform
  - Texture combine
- Break large problems into smaller stages
  - Multipass and Render-to-Texture

Further Information

OpenGL 1.x

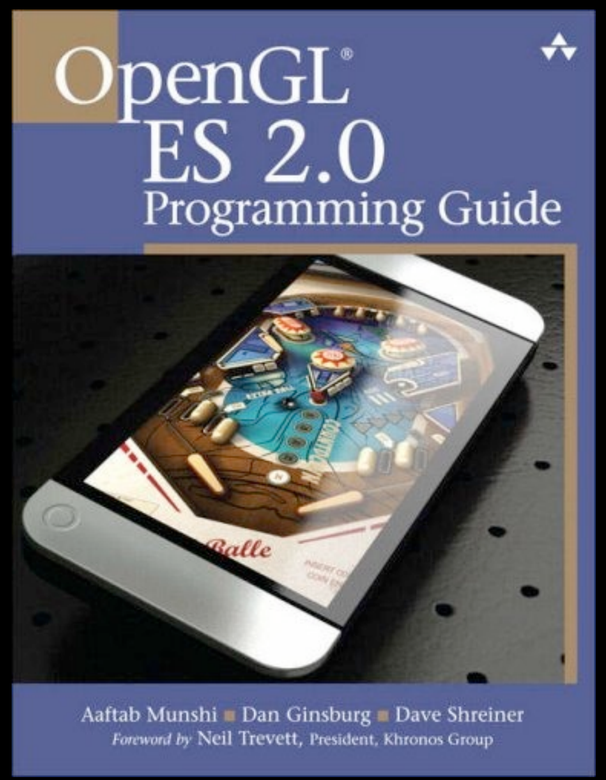
SIGGRAPH 1996-2000: Programming with OpenGL: Advanced Techniques



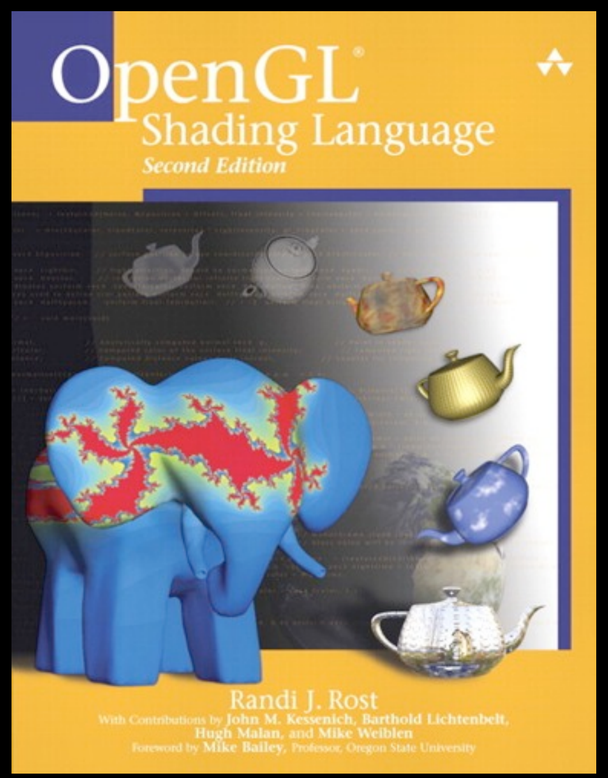
http://www.opengl.org/code/detail/siggraph_advanced_techniques/

Further Information

OpenGL ES 2.0 Programming Guide (gold) book



OpenGL Shading Language (orange) book



Related Sessions

OpenGL ES Tuning and Techniques	Presidio Wednesday 5:00PM

Labs

OpenGL ES Lab	Graphics and Media Lab A Thursday 12:00PM
Game Development for iPhone	iPhone Lab C Thursday 9:00AM

More Information

Allan Schaffer
Graphics Technology Evangelist
aschaffer@apple.com

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09